```
     (FILE 'USPAT' ENTERED AT 16:29:57 ON 28 JAN 1999)
L1          123 S BROKER# AND INSTITUTION# OR BANK# AND AGENT# AND DEALER#
L2           29 S SECURITIES TRADING
L3        26953 S COMPUTER# AND DATA BASE# OR DATABASE#
L4            6 S L1 AND L2 AND L3
```

1. 5,710,889, Jan. 20, 1998, Interface device for electronically integrating global financial services; Barry Alan Clark, et al., 345/344; 235/379, 380; 705/35, 39, 42 [IMAGE AVAILABLE]

2. 5,497,317, Mar. 5, 1996, Device and method for improving the speed and reliability of security trade settlements; John G. Hawkins, et al., 705/37 [IMAGE AVAILABLE]

3. 5,297,031, Mar. 22, 1994, Method and apparatus for order management by market **brokers**; Burton J. Gutterman, et al., 705/37 [IMAGE AVAILABLE]

4. 5,270,922, Dec. 14, 1993, System for distributing, processing and displaying financial information; Gerard M. Higgins, 705/37; 340/825.26 [IMAGE AVAILABLE]

5. 5,243,331, Sep. 7, 1993, Keypad for **computer** system; Robert G. McCausland, et al., 345/172; 340/825.26 [IMAGE AVAILABLE]

6. 5,101,353, Mar. 31, 1992, Automated system for providing liquidity to securities markets; William A. Lupien, et al., 705/37; 340/825.26, 825.27 [IMAGE AVAILABLE]

```
File  77:Conference Paper[illegible]ndex 1973-2000/Jul
         (c) 2000 Cambridge Sci Abs
File  35:Dissertation Abstracts Online 1861-2000/Jul
         (c) 2000 UMI
File 583:Gale Group Globalbase(TM) 1986-2000/Aug 31
         (c) 2000 The Gale Group
File   2:INSPEC 1969-2000/Aug W4
         (c) 2000 Institution of Electrical Engineers
File  65:Inside Conferences 1993-2000/Aug W4
         (c) 2000 BLDSC all rts. reserv.
File 233:Internet & Personal Comp. Abs. 1981-2000/Aug
         (c) 2000 Info. Today Inc.
File  99:Wilson Appl. Sci & Tech Abs 1983-2000/Jul
         (c) 2000 The HW Wilson Co.

Set     Items   Description
S1      59127   (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
               CATION? OR DATA? ? OR INFORMATION?)
S2        548   S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
               NSTITUTION?)
S3      11203   (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
               TION? OR DATA? ?)
S4        152   ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
               N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5        208   ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
               (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
               DE? OR TRANSACTION?)
S6          0   S2 AND S3 AND S4
S7          6   S2 (S) (S3 OR S4)
S8          0   S5 AND S7
S9          6   S2 AND (S3 OR S4)
S10       415   S1 AND S3
S11        25   ~~S10 AND~~ (SECURIT? OR STOCK? OR BOND? OR SHARES OR OPTION?)
S12         6   S4 AND (S1 OR S5 ~~OR S3)~~  Considered
?
```

```
File 473:Financial Times Abstracts 1998-2000/Aug 31
         (c) 2000 The New York Times
File 474:New York Times Abs 1969-2000/Aug 31
         (c) 2000 The New York Times
File 475:Wall Street Journal Abs 1973-2000/Aug 31
         (c) 2000 The New York Times

Set     Items   Description
S1       2109   (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
             CATION? OR DATA? ? OR INFORMATION?)
S2         43   S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
             NSTITUTION?)
S3         83   (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
             TION? OR DATA? ?)
S4         22   ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
             N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5        291   ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
             (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
             DE? OR TRANSACTION?)
S6          0   S2 AND S3 AND S4
S7          0   S2 (S) (S3 OR S4)
S8          0   S5 AND S7
S9          0   S2 AND (S3 OR S4)
S10         3   S1 AND (S3 OR S4)   considered
?
```

```
File 256:SoftBase:Reviews,Companies&Prods. 85-2000/Jul
         (c)2000 Info.Sources Inc
File 278:Microcomputer Software Guide 2000/Aug
         (c) 2000 Reed Elsevier Inc.

Set     Items   Description
S1       2591   (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
             CATION? OR DATA? ? OR INFORMATION?)
S2         88   S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
             NSTITUTION?)
S3        225   (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
             TION? OR DATA? ?)
S4         15   ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
             N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5          6   ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
             (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
             DE? OR TRANSACTION?)
S6          0   S2 AND S3 AND S4
S7          1   S2 (S) (S3 OR S4)
S8          0   S5 AND S7
S9         12   S1 AND (S3 OR S4)
S10         5   S4 AND (STOCK? OR SECURIT? OR SHARES OR BOND? OR OPTION?)  Considered
?
```

```
File 348:European Patents [illegible]78-2000/Sep W01
         (c) 2000 European Patent Office
File 349:PCT Fulltext 1983-2000/UB=20000824, UT=20000810
         (c) 2000 WIPO/MicroPat

Set     Items   Description
S1     116830   (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
             CATION? OR DATA? ? OR INFORMATION?)
S2       1682   S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
             NSTITUTION?)
S3      15782   (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
             TION? OR DATA? ?)
S4        235   ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
             N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5        108   ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
             (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
             DE? OR TRANSACTION?)
S6         13   S2 AND S3 AND S4
S7         73   S2 (S) (S3 OR S4)
S8          4   S5 AND S7
S9        479   (MATCH?) (N10) ((DATA OR INFORMATION?) (N5) (FIELD?))
S10         5   S9(S)(S2 OR S5 OR S4)
S11         3   S2(S)S4     Considered
?
```

```
File  15:ABI/Inform(R) 19[illegible]-2000/Aug 31
         (c) 2000 Bell & Howell
File   9:Business & Industry(R) Jul/1994-2000/Sep 01
         (c) 2000 Resp. DB Svcs.
File 623:Business Week 1985-2000/Aug W3
         (c) 2000 The McGraw-Hill Companies Inc
File 810:Business Wire 1986-1999/Feb 28
         (c) 1999 Business Wire
File 275:Gale Group Computer DB(TM) 1983-2000/Sep 01
         (c) 2000 The Gale Group
File 624:McGraw-Hill Publications 1985-2000/Aug 31
         (c) 2000 McGraw-Hill Co. Inc
File 813:PR Newswire 1987-1999/Apr 30
         (c) 1999 PR Newswire Association Inc
File 636:Gale Group Newsletter DB(TM) 1987-2000/Sep 01
         (c) 2000 The Gale Group
File 621:Gale Group New Prod.Annou.(R) 1985-2000/Aug 30
         (c) 2000 The Gale Group
File  16:Gale Group PROMT(R) 1990-2000/Sep 01
         (c) 2000 The Gale Group
File 160:Gale Group PROMT(R) 1972-1989
         (c) 1999 The Gale Group
File 148:Gale Group Trade & Industry DB 1976-2000/Sep 01
         (c)2000 The Gale Group
File  20:World Reporter 1997-2000/Sep 01
         (c) 2000 The Dialog Corporation plc

Set     Items   Description
S1     615690   (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
             CATION? OR DATA? ? OR INFORMATION?)
S2      17380   S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
             NSTITUTION?)
S3      50765   (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
             TION? OR DATA? ?)
S4       4063   ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
             N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5      25101   ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
             (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
             DE? OR TRANSACTION?)
S6          6   S2 AND S3 AND S4
S7        104   S2 (S) (S3 OR S4)
S8          6   S5 AND S7
S9        436   (MATCH?) (N10) ((DATA OR INFORMATION?) (N4) (FIELD?))
S10         0   S4 (S)S9
S11         0   S5 (S)S9
S12        10   S9 (S) (S1 OR S2)  Considered
?
```

```
File  15:ABI/Inform(R) 1[illegible]-2000/Aug 31
         (c) 2000 Bell & Howell
File   9:Business & Industry(R) Jul/1994-2000/Sep 01
         (c) 2000 Resp. DB Svcs.
File 623:Business Week 1985-2000/Aug W3
         (c) 2000 The McGraw-Hill Companies Inc
File 810:Business Wire 1986-1999/Feb 28
         (c) 1999 Business Wire
File 275:Gale Group Computer DB(TM) 1983-2000/Sep 01
         (c) 2000 The Gale Group
File 624:McGraw-Hill Publications 1985-2000/Aug 31
         (c) 2000 McGraw-Hill Co. Inc
File 813:PR Newswire 1987-1999/Apr 30
         (c) 1999 PR Newswire Association Inc
File 636:Gale Group Newsletter DB(TM) 1987-2000/Sep 01
         (c) 2000 The Gale Group
File 621:Gale Group New Prod.Annou.(R) 1985-2000/Aug 30
         (c) 2000 The Gale Group
File  16:Gale Group PROMT(R) 1990-2000/Sep 01
         (c) 2000 The Gale Group
File 160:Gale Group PROMT(R) 1972-1989
         (c) 1999 The Gale Group

Set     Items   Description
S1     415457    (TRANSMIT? OR TRANSFER? OR SEND? OR RECEIV?) (N4) (COMMUNI-
             CATION? OR DATA? ? OR INFORMATION?)
S2      11160    S1 (N25)(TRADER? OR STOCKBROKER? OR DEALER? OR AGENT? OR I-
             NSTITUTION?)
S3      31845    (MATCH?) (N10) (COMMUNICATION? OR CONFIRMATION? OR INFORMA-
             TION? OR DATA? ?)
S4       2541    ((POST?) (N4) (TRAD? OR TRANSACTION?)) (N15) (COMMUNICATIO-
             N? OR DATA OR SETTLEMENT? OR CONFIRMATION? OR INFORMATION?)
S5      14791    ((CONFIRM? OR APPROV? OR ACCEPT? OR SETTL? OR RATIF?) (N5)
             (SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?))(N10) (TRA-
             DE? OR TRANSACTION?)
S6          4    S2 AND S3 AND S4
S7         72    S2 (S) (S3 OR S4)
S8          5    S5 AND S7
S9         66    S7 NOT (S6 OR S8)
S10        50    RD (unique items)
S11        34    S10 NOT PY=1998:2000
S12         3    RD S6 (unique items)
S13         4    RD S8 (unique items)   Considered
?
```

```
File 351:DERWENT WPI 1963-1999/UD=9925;UP=9925;UM=9925
         (c)1999 Derwent Info Ltd
File 344:Chinese Patents ABS Apr 1985-1999/Jun
         (c) 1999 European Patent Office
File 347:JAPIO Oct 1976-1999/Feb.(UPDATED 990603)
         (c) 1999 JPO & JAPIO

Set     Items   Description
S1         69   (TRADER? OR STOCKBROKER? OR STOCK()DEALER? OR BROKER? OR I-
             NSTITUTION? OR AGENT?) (N40) ((TRANSMIT? OR TRANSFER? OR SEND?
              OR RECEIV?)(N3) (COMMUNICATION?))
S2        887   MATCH?(N5)COMMUNICATION?
S3          1   POST?(N4)TRADE?(N4)COMMUNICATION?
S4          3   (CONFIRM? OR APPROV? OR ACCEPT? OR SETTLE? OR RATIF?) (N10-
             )((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION?) (N10) -
             (TRAD?))
S5          0   S1 AND S4
S6          1   S1 AND S2
S7          2   S1 AND (SECURITIES? OR STOCK? OR SHARES OR BOND? OR OPTION-
             ?)
S8          7   S1 AND (COMPUTER?(N5)SYSTEM?)
S9          1   S8 NOT AGENT?
S10         7   S1 NOT (AGENT? OR INSTITUTION?)
?
```

```
File 278:Microcomputer Software Guide 1999/Jun
         (c) 1999 Reed Elsevier Inc.
File 256:SoftBase:Reviews,Companies&Prods. 85-1999/Jun
         (c)1999 Info.Sources Inc

Set     Items   Description
S1         10   (TRADER? OR STOCKBROKER? OR STOCK()DEALER? OR BROKER? OR I-
              NSTITUTION? OR AGENT?) (N40) ((TRANSMIT? OR TRANSFER? OR SEND?
               OR RECEIV?)(N3) (COMMUNICATION?))
S2          2   MATCH?(N5)COMMUNICATION?
S3          0   POST?(N4)TRADE?(N4)COMMUNICATION?
S4         12   (CONFIRM? OR APPROV? OR ACCEPT? OR SETTLE? OR RATIF?) (N10-
              )((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION?) (N10) -
              (TRAD?))
S5          0   S1 AND S4
S6          0   S1 AND S2
S7          0   TRADESUITE
S8          7   E3-E7
```

```
File 621:New Prod.Annou.(R) 1985-1999/Jul 09
         (c) 1999 The Gale Group
File 636:Newsletter DB(TM) 1987-1999/Jul 09
         (c) 1999 The Gale Group
File  16:PROMT(R) 1972-1999/Jul 09
         (c) 1999 The Gale Group
File 148:Trade & Industry Database 1976-1999/Jul 09
         (c) 1999 The Gale Group
File 624:McGraw-Hill Publications 1985-1999/Jul 08
         (c) 1999 McGraw-Hill Co. Inc
File 484:Periodical Abstracts Plustext 1986-1999/Jun W3
         (c) 1999 UMI
File 141:Readers Guide 1983-1999/May
         (c) 1999 The HW Wilson Co
File 370:Science 1996-1999/May W3
         (c) 1999 AAAS
File 696:DIALOG Telecom. Newsletters 1995-1999/Jul 09
         (c) 1999 The Dialog Corp.
File 553:Wilson Bus. Abs. FullText 1982-1999/Jun
         (c) 1999 The HW Wilson Co

Set     Items   Description
S1        550   (TRADER? OR STOCKBROKER? OR STOCK()DEALER? OR BROKER? OR I-
             NSTITUTION? OR AGENT?) (N40) ((TRANSMIT? OR TRANSFER? OR SEND?
              OR RECEIV?)(N3) (COMMUNICATION?))
S2       1848   MATCH?(N5)COMMUNICATION?
S3        221   POST?(N4)TRADE?(N4)COMMUNICATION?
S4      14859   (CONFIRM? OR APPROV? OR ACCEPT? OR SETTLE? OR RATIF?) (N10-
             )((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION?) (N10) -
             (TRAD?))
S5          3   S1 AND S4
S6          1   S1 AND S2
S7         13   S1 (N30) ((SECURIT? OR STOCK? OR SHARES? OR BOND? OR OPTIO-
             N?) (N10) (TRAD?))
S8          0   S2 AND S3
S9          0   S3 AND S1
S10        60   S1(S) (COMPUTER? OR MICROPROCESSOR? OR MICROCOMPUTER?)
S11         0   S10 AND S4
S12         0   S10 AND (S2 OR S3)
S13         3   RD S5 (unique items)
S14         1   RD S6 (unique items)
S15        10   RD S7 (unique items)
?
```

full text

```
File  15:ABI/INFORM(R) 1971-1999/Jul 09
         (c) 1999 UMI
File   9:Business & Industry(R) Jul 1994-1999/Jul 09
         (c) 1999 Resp. DB Svcs.
File  13:BAMP 1999/Jun W4
         (c) 1999 Resp. DB Svcs.
File 647:CMP  Computer Fulltext 1988-1999/Jun W4
         (c) 1999 CMP
File 674:Computer News Fulltext 1989-1999/Jun W4
         (c) 1999 IDG Communications
File  98:General Sci Abs/Full-Text 1984-1999/May
         (c) 1999 The HW Wilson Co.
File 275:Computer Database(TM) 1983-1999/Jul 09
         (c) 1999 The Gale Group
File  47:Magazine Database(TM) 1959-1999/Jul 09
         (c) 1999 The Gale group

Set     Items   Description
S1        264   (TRADER? OR STOCKBROKER? OR STOCK()DEALER? OR BROKER? OR I-
             NSTITUTION? OR AGENT?) (N40) ((TRANSMIT? OR TRANSFER? OR SEND?
              OR RECEIV?)(N3) (COMMUNICATION?))
S2        885   MATCH?(N5)COMMUNICATION?
S3         36   POST?(N4)TRADE?(N4)COMMUNICATION?
S4       3347   (CONFIRM? OR APPROV? OR ACCEPT? OR SETTLE? OR RATIF?) (N10-
             )((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION?) (N10) -
             (TRAD?))
S5          2   S1 AND S4
S6          1   S1 AND S2
S7          7   S1 (S) ((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION-
             ?) (N10) (TRAD?))
S8         10   S3 AND MATCH?
?
```

Full-text files

```
File   8:Ei Compendex(R) 1970-1999/Jul W1
         (c) 1999 Engineering Info. Inc.
File  77:Conference Papers Index 1973-1999/Jul
         (c) 1999 Cambridge Sci Abs
File 238:Abs. in New Tech & Eng. 1981-1999/Jun
         (c) 1999 Reed-Elsevier (UK) Ltd.
File  35:Dissertation Abstracts Online 1861-1999/Jul
         (c) 1999 UMI
File  65:Inside Conferences  1993-1999/May W5
         (c) 1999 BLDSC all rts. reserv.
File   2:INSPEC 1969-1999/Jun W4
         (c) 1999 Institution of Electrical Engineers
File  94:JICST-EPlus 1985-1999/Mar W4
         (c)1999 Japan Science and Tech Corp(JST)
File 233:Microcomputer Abstracts 1974-1999/Jul
         (c) 1999 Information Today Incl.
File 144:PASCAL 1973-1999/JUN
         (c) 1999 INIST/CNRS
File   6:NTIS 64-1999/Aug W2
         Comp&distr 1998 NTIS, Intl Copyright All Righ
File  34:SciSearch(R) Cited Ref Sci 1990-1999/Jul W1
         (c) 1999 Inst for Sci Info
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File  99:Wilson Appl. Sci & Tech Abs 1983-1999/May
         (c) 1999 The HW Wilson Co.

Set     Items   Description
S1         79   (TRADER? OR STOCKBROKER? OR STOCK()DEALER? OR BROKER? OR I-
              NSTITUTION? OR AGENT?) (N40) ((TRANSMIT? OR TRANSFER? OR SEND?
               OR RECEIV?)(N3) (COMMUNICATION?))
S2        972   MATCH?(N5)COMMUNICATION?
S3          6   POST?(N4)TRADE?(N4)COMMUNICATION?
S4        126   (CONFIRM? OR APPROV? OR ACCEPT? OR SETTLE? OR RATIF?) (N10-
              )((SECURITIE? OR STOCK? OR SHARES OR BOND? OR OPTION?) (N10) -
              (TRAD?))
S5          0   S1 AND S4
S6          0   S1 AND S2
S7          0   S1 AND ((SECURIT? OR STOCK? OR SHARES OR BOND? OR OPTION?)
              (N10) (TRAD?))
?
```

Bibliographie

01295251 SUPPLIER NUMBER: 07589659 (THIS IS THE FULL TEXT)
An object-oriented requirements specification method.
Bailin, Sidney C.
Communications of the ACM, v32, n5, p608(16)
May, 1989
ISSN: 0001-0782 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 8770 LINE COUNT: 00694

ABSTRACT: Object-oriented software requirements analysis is approached with methodology different from the more conventional structured analysis method. This alternative methodology is detailed through the use of parallel processes of decomposing objects and allocating functions. The new method is designed to flow smoothly into design by object diagrams and from there into programming with Ada or other high level languages. The primary concept is of entity rather than process. The specification method has seven steps: identify key problem domain entities; distinguish between active and passive entities; establish data flow between actives entities; decompose entities into sub-entities and - or functions; check for new entities; group functions under new entities; and assign new entities to appropriate domains.
TEXT:

AN OBJECT-ORIENTED REQUIREMENTS SPECIFICATION METHOD This article describes a method of analyzing requirements for object-oriented software. The method is intended to flow smoothly into design by object diagrams, [15], and from there into programming with Ada or another high-level language. Our method is intended to serve as an alternative to structured analysis when the use of object-oriented design is foreseen.

We assume that the analyst who is using this method has a textual statement of requirements for a system available. Ideally, the analyst would already have distilled the requirements statement into a database of discreet, traceable requirements. This is not necessary for the method we describe here, but it is highly advisable. The steps we describe are part of a method for understanding and articulating the implications of the original requirements (whether in text or database form).

Structured analysis is the method most widely used for this purpose. So why not use structured analysis instead of inventing a new method? To answer this we must reflect on what structured analysis offers us. Structured analysis is a method of articulating functional requirements. Some--perhaps most--of the initial set of requirements for a system will be functional, i.e., they will state that the system must accept certain inputs and deliver certain outputs. In structured analysis we attempt to clarify what such statements really mean. Processes are introduced to represent transformations of inputs to outputs. To ensure that all parties involved (customer, end-user, developer, etc.) have a common understanding of what is to be accomplished by the transformations, we decompose the processes. For each transformation of inputs to outputs we ask, What are the steps involved in performing this transformation? Each step becomes a subprocess of the higher level process. Thus, built into structured analysis is the following principle of aggregation:

Functions are grouped together if they are constituent steps in the execution of a higher level function.

The constituent steps may operate on entirely different data abstractions (we will see this in an example in the next section). If this is the case, then from an object-oriented viewpoint the subfunctions belong in different objects. The object-oriented principle of aggregation refers, unlike structured analysis, to the underlying data:

Functions are grouped together if they operate on the same data abstraction.

In an object-oriented design, functions executed in sequence can reside in different objects. A chain of messages between objects effects the sequence of operations.

Because of the difference in aggregation principles, proceeding from a structured analysis to an object-oriented design can be awkward. Since the criteria for grouping functions are different in the two methods, the transition from one to the other may require significant recasting of the data flow diagrams. This is a laborious process, which can be avoided by assuming an object-oriented viewpoint during the analysis phase.

In our method the primary concept is that of entity rather than process. A process in structured analysis may be a valid entity but the connotation of entity is different. Our emphasis is not on the transformation of inputs to outputs, but on the underlying "content" of the entity. We can think of this content in several ways:

* Data structures that define the entity
* The underlying state of a process as it evolves in time
* That aspect of a process that is persistent across repeated execution cycles

Complementing our notion of entity is the notion of function. A function is purely a transformation of inputs to outputs. It has no underlying state that it remembers across successive invocations.

Through the distinction between entities and functions, our methodology achieves a precision that is lacking in structured analysis. Every process in a structured analysis is, implicitly, either an entity or a function--but because the emphasis is on transforming inputs to outputs, we lack a means to focus on the persistent aspect of entities. That is why the object-oriented principle of aggregation is not articulated in structured analysis.

In our method we retain the data flow diagram representation of inputs and outputs. The idea of specifying data flow between entities is taken from the Object Dataflow Diagrams of Stark and Seidewitz. We use the term entity data flow diagrams for the same concept. EDFDs are just like conventional DFDs except that the nodes fall into two categories: entities and functions. Every function must occur in the context of an entity, i.e., it must be performed by or act on the entity (see Figure 1, which uses a notation derived from Object Diagrams).

We distinguish between active and passive entities. Active entities are represented as diagram nodes, while passive entities appear as data flows or data stores. Entity for us is synonymous with object (we tend to use entity during the analysis phase and object during design).

Lower level EDFDs decompose each active entity either into subentities or into functions performed by the entity (or into some combination of the two). Functions may be decomposed into subfunctions. The resulting EDFD hierarchy consists of an upper hierarchy of entities and subentities, a wavy line of lowest-level entities, and a hierarchy of functions and subfunctions below each lowest-level entity. This is illustrated in Figure 2.

An object-oriented specification consists of a **hierarchy** of EDFDs and a set of entity relationship diagrams. The **entity relationship model** serves as a context for the specification. The E-R model provides both users and builders with a comprehensive view of the problem domain--and, possibly, some technology domains that are known to be part of the context of the system. (Purists may reject the presence of technological considerations in a requirements analysis. We refer them to the discussion of entity domains and the discussion of analysis vs. design, later in the article, for a defense of our approach. A general discussion of the early stages of software development may be found in [10]).

DETAILED DESCRIPTION OF THE METHOD

The specification method consists of seven steps:

(1) Identify key problem-domain entities.
(2) Distinguish between active and passive entities.
(3) Establish data flow between active entities.
(4) Decompose entities (or functions) into sub-entities and/or functions.
(5) Check for new entities.
(6) Group functions under new entities.
(7) Assign new entities to appropriate domains.

Although steps one through three are, in principle, performed only once, we may backtrack at any time to reconsider these steps. Steps four through seven should be iterated until the desired level of detail is reached.

Identify Key Problem-Domain Entities

Identification of the right problem-domain entities is crucial to the development of a stable design. There is no hard and fast way to define what is (and what is not) a problem-domain entity. Part of the analysis process is to assess the consequences of including (or not including) potential problem-domain entities. Ask the following questions:

* Is every known functional requirement met by one of the entities?
* Is the internal state of the system adequately represented by the states of the entities?

An excellent discussion of these and related issues may be found in the first chapter of Jackson's book System Development.

A heuristic procedure for identifying the problem-domain entities is suggested by Grady Booch in the first edition of his book, Software Engineering with Ada. The procedure is to create a list of the key nouns and noun phrases from the original (textual) statement of requirements. Such a list may serve as a first approximation to the problem-domain entities. We must take this heuristic with a grain of salt, however. Booch even omitted the procedure from the second edition of his book. In a typical requirements specification there will be several nouns that are completely irrelevant to the problem-domain entities. We must be highly selective when examining the specification for key nouns.

A more realistic variation on this approach would be to locate the nouns and noun phrases in the requirements database (if there is one). A requirements database distills the original textual statement into a set of traceable requirements, in a form that is less verbose than the original statement. In performing such distillation, engineering judgment and selection are applied to the original statement. The database therefore offers a better chance of identifying truly significant nouns.

A still more manageable approach draws on the process-orientation of structured analysis and helps to clarify the relationship between that method and ours. The typical process name in structured analysis will take the form. action_object.

More often than not, the objects in such names refer to problem-domain entities. We have found it useful to draw structured analysis DFDs merely to extract the nouns from the process names. We do this repeatedly, throughout the seven steps of our method. The structured analysis DFDs are not part of the object-oriented specification--they serve only as intermediate representations--but they are an effective way of identifying entities.

We draw an Entity-Relationship Diagram to record the problem-domain entities and their interrelationships. Figure 3 shows an ERD for an automated banking application, which we will use as an example. As recommended by Stark and Seidewitz [16] we may also create an entity dictionary, which is analogous to the data dictionary in structured analysis.

Distinguish Between Active and Passive Entities

The most natural distinction between active and passive entities is in the following definition.

An active entity is one that operates on inputs to produce outputs. A passive entity is one that is acted upon.

This is not the definition we use. The need for a slighlty different definition arises from the following requirements:

* Active entities as should appear as processes, passive entities as data flows
* Every function must be performed by some entity

The first of these requirements provides us with a straightforward semantics for EDFDs. The second requirement is our insurance that the specification will be objected-oriented.

The problem is that passive entities may consist of more than just data: they may provide a set of primitive functions for accessing the data. Abstract Data Types (ADTs) such as lists and stacks are the best-known examples. If we represent such entities as data flows, how do we represent the built-in functions? More precisely, how do we represent both data and functions as part of one and the same entity?

There is no problem if the functions are sufficiently low-level. Operations on an ADT my be primitive enough to warrant not including them in requirements analysis--at least for large systems. Other operations on passive entities (using the definition just given) may be essential to specifying requirements though. We might, for example, want to sort a stream of elements input from a file. In structured analysis this is simple. We represent the sort as a process and the input stream as a data flow (Figure 4). In OOS the process sort is in fact a function. It operates on a list elements with a built-in partial ordering relation, and returns a sorted list. From an object-oriented viewpoint, sort is an operation provided by an entity whose underlying content is the list of elements (Figure 5, which uses an object diagram rather than EDFD notation).

The difference between Figures 4 and 5 illustrates in as concise a manner as possible the difference between structured analysis and OOS. In Figure 4, the operation (sort) and the data stream (being sorted) are not packaged together. Sort appears as a standalone function. In Figure 5, the data and operations are packaged together according to the object-oriented principle of aggregation.

In order to adhere to this principle, we need to represent entities such as sortable list, as diagram nodes, i.e., active entities, rather than as data flows. We choose to treat them as active entities despite the fact that they perform operations on themselves. We consider them to be active because they are doing something important (whether to themselves or to other entities). The following definition expresses this approach:

An active entity is an entity whose functions we want to consider in the analysis phase. A passive entity is one whose functions we do not want to consider until the design phase.

For example, the active entities in Figure 3 are Customer and Account. Customer is obviously active. Account might be considered passive with respect to our original definition. To specify a banking system, however, we have to consider the functions by which accounts are maintained. This leads us to make Account an active entity in terms of our revised definition.

Establish Data Flow Between Active Entities

In this step we draw a top-level entity data flow diagram (EDFD). Each active entity is the ERD becomes a process in an EDFD. Passive entities should appear as data flows or data stores.

The level-0 EDFD should contain the top-level active entities. The ERD does not explicitly distinguish between top-level and lower-level entities, so if we have not already made this distinction we will have to do so now. Typically, the ERD may show relationships with names like "contains" or "includes." This is an indication that a contained or included entity should appear in some lower-level EDFD, rather than at level 0.

The level-0 EDFD should contain only entities. This will ensure that every function is provided by some entity. Lower-level EDFDs may contain both entities and functions. We differentiate between entities and functions by placing brackets around the name of an entity as shown in Figure 6a. This is meant to be suggestive of the PAMELA notation for Abstract Data Types [8]. (In PAMELA, to indicate that a process is an ADT one encloses the name of the ADT in a rectagle.)

Figure 6a illustrates a common phenomenon in OOS. The entity "Customer" also appears as an external. This indicates that the entity is a system (or software) model of an external entity. The model serves as the interface to the external entity. This principle is articulated as an explicit step in the JSD methodology [12]. We have found that it occurs naturally in the object-oriented specification process.

We have defined some rules for ensuring mutual consistency between the E-R and data flow models. One rule states that the entities in the two models must correspond one-to-one. Another rule states that any relationship in the E-R model must be manifested, in some manner, in the data flow model. This rule is somewhat conjectural. We are not sure it is true, but we are interested in understanding the extent to which it is not true (i.e., what types of relationships would not be manifested in a data flow model?) The section on pre-defined relational types addresses this further.

There are, it seems, only a few ways for a relationship to be manifested in a data flow model. Containment is one way. An active or passive entity (i.e., a process or data flow) may be contained in a higher-level active entity (process). A passive entity may also, in principle, be contained in a higher-level passive entity, but we have no way of showing this in an EDFD. An entity dictionary could be used for this purpose.

Data Flow is another way to manifest relationships in an EDFD. If a passive entity is represented as a data flow into or out of an active entity, there is clearly some relationship between them. If two active entities are connected by a data flow then there is also some relationship present.

We have developed a program that executes these checks automatically. A sample run of the rule checker on Figures 3 and 6b yields the following messages:

The entity: Checks in the ERD: Figure3 does not appear in the DFD hierarchy: Figure6a.

Entity Account is a process, and entity Transactions is a dataflow, and Account and Transactions are related, but Transactions is not contained in, nor is it an input nor an output of Account.

To eliminate these errors we revise the top-level EDFD to look like Figure 6b.

The conjectural status of our relationship rule is not our reason for not using the rule checker. The purpose of the rule checker is to ensure that we have adequately addressed the consistency of the E-R and data flow models. If relationships in the E-R model are not represented in the data flow model, the rule checker will report this; we are then free to decide that the situation is acceptable or not, as the case may be.

Decompose Entities (or Functions) into Subentities and/or Functions

Steps four through six form the heart of the OOS method. When performing step four the first time, we decompose the top-level EDFD. Nodes in this diagram are all active entities. We decompose each active entity into subentities and/or functions. The subentities are lower-level entities out of which the higher-level entity is composed. The functions (if there are any) are performed by the entity being decomposed. Figure 7, for example, shows a diagram that decomposes the Account entity of Figure 6b.

In subsequent passes through this step we decompose both entities and functions. An entity may be decomposed into subentities and/or functions. Funtions, however, may only be decomposed into subfunctions.

To identify functions we consider what the entity does. As in step one, we ask whether every known functional requirement can be met by one or more of the functions identified. We may also apply the same sort of heuristics that we applied in step one. For example, we might look for verbs and verb phrases in the initial (textual) requirements specification, or in the requirements database (if this exists). Some of the verbs may refer to functions that the system is supposed to perform. We emphasize, though, that judgment is required in selecting only significant verbs. The action_object heuristic also applies. Prior to this point we may have applied this heuristic to identifying entities. We drew structured analysis DFDs and extracted the object from each process name. We may now apply the same heuristic to identifying functions. In this case the action becomes the function.

Check For New Entities

At each stage of decomposition, we must stop and ask whether any new entities are implied by the functions we have introduced. The action_object heuristic again applies. For each function introduced through this heuristic in step four, we now consider the corresponding entity that was the object of the action. More generally, regardless of the method used in step four, we can put all new function names into the form action_object, and examine these names for objects not previously identified.

For example, in Figure 7 we find a function with the label Transfer_Funds_to_Other_Bank We see that Other_People's_Banks appears as an external but not as an entity within our system. We should consider, then, whether Other_People's_Banks should have an internal interface entity just as Customer does. We should also consider whether Funds should be identified as an entity.

Data stores and data flows may have been introduced through step four. These are potential passive entities. We examine each new flow or store and ask whether it is significant enough to include in the E-R model. Some new flows or stores may represent transformed versions of passive entities already identified, and in such cases we do not add a new entity. Exceptions are discussed in the section on identification of passive entities under successive transformations.

An active entity must be added unless it is already part of the specification. Since we have been working with the data flow model, we first add the entity to an appropriate EDFD. (We will add it to the E-R model as well, in step seven.) Deciding where in the EDFDs to place the new entity is not always a trivial task. One alternative is to replace a function by a new entity. The function might then be placed one level down, below the new entity.

In our example, although Other_Banks was identified through the function Transfer_Funds_to_Other_Bank, which is provided by Account, we would not place Other_Banks under Account. This would not make any sense. Instead, we would create Other_Banks, a new top-level entity, which would interface with the external entity of the same name. The resulting top-level EDFD is shown in Figure 8.

We do not offer any guidelines for placing new active entities except to note that the diagrams may have to be reorganized. It is in this step and the next that we consider such issues as strength and coupling, information hiding and visibility, and the quality of entity abstractions.

Reorganization of DFDs, as we noted in the beginning of this article, is the precise difficulty in proceeding from a structured analysis to an object-oriented design. We now see that our OOS method does not eliminate this task. It simply distributes it over the course of the requirements analysis so that an object orientation is retained throughout. The same amount of work may, in the end, be necessary (we are not sure of this), but eh force required is less because the structure is not allowed to become too bad (i.e., too functional) before being corrected.

Group Functions Under New Entities

Since we introduced new entities in step five, we must now identify all the functions performed by (or on) the new entities. This might mean making a passive entity active.

Checking for additional functions is a way of ensuring a faithful representation of entities. We call this feature functional completeness: it is characteristic of the object-oriented approach. It provides object-oriented specifications (and subsequent designs) with stability under evolving functional requirements. By including all functions performed by an entity--whether they are immediately required or not--we increase the chances of being able to respond to future functional enhancements with minimal respecification and redesign. We also increase the probability of developing reusable components.

Besides checking for new functions, we need to examine the EDFDs and reassess the placement of functions already present. We introduced new entities through step five. These new entities may now serve as the most natural providers of functions that were previously present (and provided by other entities). In our example, we conclude that the functions * Receive_Incoming_Checks * Return_No_Good_Checks and * Transfer_Funds_to_ Other_Bank, which were originally provided by the entity Account, should now be moved under Other_Banks. We then consider whether there are other functions or atrribute data that should be part of this entity. Figure 9 shows the resulting decomposition of Other_Banks.

The analyst must be open to reorganizing the EDFDs in this step. It is this interaction of entities and functions (i.e., entities providing new functions, functions suggesting new entities, and the consequent recasting of diagrams) that constitutes the life-blood of our method and that justifies our inclusion of functinal decomposition within an object-oriented approach.

Assign New Entities To Appropriate Domains

This last step is intended to help in managing the complexity of proliferating entities. We recommend assigning each new entity (introduced through step five) to some domain. The domain may be new. In figure 9, for example, further analysis of the external interface may lead us to specify standard message formats from the domain of Electronic Funds Transfer (EFT). Having established that EFT is an existing technology to be used in the application, the data flows between the internal entity Other_Banks and its external counterpart may be assigned to the EFT domain.

If the E-R model is becoming complex, we may want to represent each domain in a separate diagram. This will prevent the original ERD (which represents the application or problem domain) from growing too large to visualize. The set of ERDs, taken together, provides a comprehensible view of the entities of the system.

We find it helpful to consider different levels of entity domains. For example, the following list describes a hierarchy of types of domains. This is a kind of "virtual-machine" hierarchy, in so far as higher-level domains use the domains at lower levels: (1) Application Domains (2) Technology Domains (3) Computer Science Domains (4) Execution Models (5) Execution Domains

This hierarchy of domain levels was proposed by Ira Baxter of the University of California at Irvine--see references [4] and [5] for a discussion of the underlying ideas. Baxter's work occurs in the context of the Draco methodology, which is discussed by James Neighbors in [13] and [14].

During the specification phase we are most likely interested only in the application domain and, possibly, some technology domains such as communications, database management, signal processing, and graphics. As we approach the design phase we may begin to address classes of algorithms and implementation approaches. Computer Science domains provide the concepts for expressing and analyzing such issues. For those who might argue that only the problem domain is relevant to requirements analysis, we refer to the later section on analysis or design.

POSSIBLE EXTENSIONS OF THE METHOD

In this section we discuss some aspects of requirements definition that we cannot, as yet, represent in an object-oriented specification. We are continuing to experiment with extensions and variations of the ERD and EDFD representations in order to accommodate these aspects of system requirements. A discussion of various representation mechanisms can be found in [7].

Identifying Passive Entities under Successive Transformations

In the pipeline paradigm of data processing, a stream of elements is input from some source, passed through a series of transformations and filters, and fed out to some sink. Extensions of this paradigm allow for more than one source and more than one sink. It seems safe to say that this paradigm, in its extended form, underlies the thinking that led to structured analysis. Structured analysis, however, has also been applied to systems not adequately described by the (extended) pipeline paradigm. Structured analysis has encouraged this view of systems, and it has proven to be a useful view even when other views are necessary.

One helpful feature of the pipeline view is that we can regard a data element as undergoing successive transformations while still retaining its identity. Unfortunately, the rules for Data Flow Diagrams discourage representing this continuity explicitly. Each data flow must have a unique name. Usually it is just a visual feature of the diagram that allows us to infer the identity of a data element as it passes from process to process, i.e., we can see the data passing through successive processes.

In structured analysis, seeing this is enough. In object-oriented specification we correlate data flows with entities in the E-R model, and it is here that the naming rules pose a problem. As the method now stands we must select one out of possibly several flows (related to each other by one or more transformations), and decide that the selected flow, and not its transformed versions, corresponds to the passive entity. It would be helpful to be able to give all these flows the same name. This is discouraged in structured analysis because it suggests that the processes, which input data elements and then output the same data elements, are not doing anything.

Another way to identify passive entities under successive transformations is to annotate the data flows. This amounts to giving them a second name which would not be subject to the uniqueness rule (i.e., more than one data flow could have the same second name). We find this approach less than inspiring. It does not make use of the expressive power of the EDFDs themselves. If an automated drawing tool is used, annotation may or may not be possible. Even if supported by a tool, annotations may not be visible while editing the diagrams. Providing data flows with aliases does not make the identity of transformed data flows an integral part of the method.

It may be that a representation other than EDFDs is needed. This possibility is further suggested by a question that emcompasses, but is broader than, the identity of passive entities under transformation. The broader issue concerns end-to-end service definition, and will now be discussed.

End-to-End Service Definition

The concept of service is found in many operational concepts. There are probably as many definitions of service as there are of operational concept, but common to all definitions is an end-user orientation.

The concept of service can fit nicely into an object-oriented framework. For example, services are sometimes defined as groups of related functions. This is not a bad definition of objects, although for objects, it is usually some data abstraction that serves to relate the functions with each other. One way to incorporate the concept of services into OOS is, therefore, to identify services with entities. The name of the service often suggests the entity. In our banking example we might refer to an automated teller service, a customer support service, a loan service, a new account service, and others. As entities we could identify the automated teller, the customer, loans, and accounts.

Conversely, object-oriented principles imply the concept of services. As a black box with a well-defined interface, an object is essentially a service access point (to borrow the terminology from open systems interconnection).

This approach identifies the service with its user interface (the user might be the end-user or another service or entity). There are, however, some drawbacks to doing this. Identifying the service with its user interface complicates the task of clarifying what the service does for the user. Much of what the service does occurs behind the interface, and may not be performed by the entity that provides the interface. This is a consequences of the object-oriented principle of aggregation. Execution of a task may require a chain of messages between objects as mentioned earlier. In order to express end-to-end response time requirements we need to refer to such chains explicitly.

Our representation provides no way to do this. In entity data flow diagrams we are limited to tracing data flows through a sequence of processing steps. The sequence itself is not explicitly represented. Referring to the time between the call to an interface function and its return is not sufficient. Through asynchronous processing, the output of a service request may be delivered after the interface function has returned as displayed in Figure 10.

A formalism such as R-Nets (part of M. Alford's SREM or Distributed Computing Design System methodology) can be sued to specify end-to-end processing of service requests [1]. CTA uses a variant of this formalism, which we call composition graphs, to define end-user operational requirements. Figure 11 shows a composition graph corresponding tothe EDFD in Figure 7. These formalisms are especially useful for specifying end-to-end response time requirements and representing the budgets assigned to individual processing steps.

Enriching the Entity-Relationship Model

Our method uses entity relationship diagrams to provide a comprehensible picture of the problem domain and, perhaps, of some technology domains that are assumed as a context for the problem. Most of the analysis process focuses on the EDFD representation. We are considering ways to enrich the E-R model so that it plays a more substantive role in the analysis process. The enrichments we will describe would provide greater isomorphism between the E-R model and the data flow and object diagram models.

Pre-Defined Relationship Types

In Section 2.3 we conjectured that relationships in the E-R model should be manifested somehow in the EDFDs. We also described two ways in which relationships may be manifested in the EDFDs, i.e., through containment or with data flow. Reserving names for these types of relationships would give the ERDs more precise semantics, and would enable us to check more precisely the consistency of the E-R and data flow models.

If our conjecture is false, there is all the more reason to reserve names for these relationship types. By restricting the requirement for manifestation in an EDFD to these relationship types, we would remove the rule's conjectural status. We could also explore the possiblity of other pre-defined relationship types, (e.g., uses, inputs, outputs). The intention would not be to restrict all relationships to one of these types, but rather to make these pre-defined types available and thereby enrich the semantics of the ERDs.

An alternative to reserved names is to recognize grammatical classes of relationships. For example, we could distinguish between transtive verbs (such as updates) and intransitive predicates (such as is part of). It might be possible to define rules that govern how much relationships are manifested in the EDFDs based on their grammatical class. For example, we might be able to infer whether an entity is active or passive based on the types of relationships it has with other entities.

Functions as Attributes of Entities

Functions immediately below the wavy line in Figure 2 are the operations provided by entities. The entitles are above the wavy line. In Smalltalk terminology the operations are known as methods. In object diagrams, they are listed as annotations to the objects. The object's methods appear together with its visible state variables and data types as shown in Figure 12.

In the terminology of an E-R model, state variables are attributes of an entity. Since an entity (object provides functions (methods) as well as state variables, it seems reasonalb eto view these functions as attributes of the entity. We could use the standard notation for attributes to specify what functions each entity provides.

Specifying functions as attributes of entities would provide a more precise semantics for the E-R model. It would also provide another way to check for consistency between the E-R and data flow models. Every functional attribute of an entity would have to appear as a function in the EDFD that decomposes that entity. Every data-valued attribute of an entity would have to appear as a data store in the EDFD. (This last rule could also be applied to a more conventional ERD semantics in which all attributes are assumed to be data-valued.) A drawback to specifying functions as attributes is the possibility of cluttering the ERDs.

Relationships Between Domains

We have spoken of a hierarchy of domains with the problem domain at the top. Under the problem domain are one or more technology domains, and under these are the Computer Science domains. A domain at any level of this hierarchy may employ entities belonging to lower-level domains, as well as entities belonging to other domains at the same level.

There are two ways in which one domain can employ another: messages and containment. Messages are the means by which one entity invokes an operation provided by another entity. Containment refers to the decomposition of an entity belonging to a higher-level domain into entities belonging to lower-level domains. For example, the problem domain entity transaction might be represented in terms of standard communications protocol data units.

In either case, the relationship between entities in different domains amounts to a relationship between the domains themselves. It might be useful to represent these relationships in a high-level ERD whose entities would be the domains themselves. The high-level diagram would provide a global picture of the domains that form the context of the problem.

It would also be useful to represent cross-domain relationships in the domain-specific ERDs. This could be accomplished by introducing "off-edge" relationships, which would connect entities in different diagram (Figure 13).

The State-Transition Model

Entities have evolving states, distinguishing them from functions. Modeling entities as state machines seems a natural way to incorporate state-transition semantics into an OOS. This is the approach taken by Hatley and Pirbhai, although they still speak in terms of processes rather than entities.

The difficult question is how to correlate state-transitions with the data flow model. The most obvious answer is to identify selected data flows with events. For example, if a data flow corresponds to a specific interrupt signal, then this is a reasonable approach.

In other contexts, however, different values of the same data flow may correspond to different events. This is the case whenever the processing logic takes conditional branches depending on the value of the input data, and the different branches result in different state updates. In some cases, ranges of values for a given data flow can be identified with different events. In such cases we can represent the state-transition logic in an EDFD by assigning a different function to handle the data in each range.

If events cannot be correlated with either data flows or ranges of data flow values, then the relationship between the two models must be left implicit. This, at any rate, is our current view. We are continuing to experiment with ideas for mapping between the two views.

ADDITIONAL ISSUES AND CLARIFICATIONS

We will address some of the questions posed when explaining the OOS method. These points are not crucial to understanding the method, but we have found that a satisfactory answer goes far in increasing people's confidence in our approach.

System or Software Requirements?

We developed the method with software in mind--specifically Ada--but we believe it applies equally well to specifying requirements of hardware/software systems. At some point the question of hardware/software allocation must be addressed. An entity may be allocated to hardware, to software, or to a hardware/software combination. Allocations to hardware and to software must be preserved under decomposition. A subentity of a hardware entity must be hardware; a subentity of a software entity must be software, and the same is true for functions. A subentity or subfunction of a combined hardware/software entity (or function) can itself be a hardware/software combination, or just hardware, or just software. This is illustrated in Figure 14.

Is the Method New?

The answer is both yes and no. Under the guise of structured analysis, engineers have been doing object-oriented specification for years! Look at the requirements specification for a system of any significant size, and see whether or not the data flow diagrams contain all the meaningless pseudo-action words that structured analysis purists warn against: * Control * Manage * Monitor * Handle and others. How many times have we and our colleagues struggled over finding more expresive action words; yet the pseudo-actions seem appropriate, and the reason is that the substance is not in the action but in the object.

Analysts have tacitly chosen to specify and decompose entities when appropriate, disguising them as structured analysis processes. It is in such situations that strength and coupling questions are addressed. We have often wondered about strength and coupling metrics for data flow diagrams, even though such considerations should really not occur in pure structured analysis. Refer back to the aggregation principle of structured analysis, which we summarized in the first part of this article. If functions are to be grouped according to their place in the execution of a higher-level process, then strength and coupling considerations are irrelevant. Again we see object-oriented principles creep into what are ostensibly functional decompositions.

Unfortunately, counting data flows--the basis of conventional strength and coupling measurements--is not necessarily the best approach to assessing strength and coupling of objects. Principles of data encapsulation and information hiding are, perhaps, more germane. This is one argument for defining a method that is patently object-oriented.

Another argument for OOS is that by applying such principles consciously, rather than by accident, a more consistent product is likely to result.

Is it Analysis or Design?

The conventional wisdom is that analysis addresses what and design addresses how. We find this insight valid but infuriating. It contains some important truth, but it ignores some subtle issues. To begin with let us observe that in structured analysis one often regroups functions. This may occur for either of two reasons: * Strength and coupling considerations * Alternative decompositions of a process

We have already acknowledged that strength and coupling considerations should not occur in pure structured analysis. Alternative decompositions are, however, a legitimate concern--even in pure structured analysis. To transform inputs <i1, i2, ...> into outputs <o1, o2, ...>

there may be several different sequences of steps that can be taken. There may be many criteria that suggest one decomposition over another, (e.g., organization of the user's institution, reusability, or extensibility).

We should not delude ourselves into thinking that structured analysis merely makes explicit what is already implicit in the original statement of requirements. Structured analysis adds content. We must therefore consider the difference between adding content in answer to what (analysis) and adding content in answer to how (design).

The difference lies in the distinction between problem and solution domains. If the additional content refers to a domain that the customer does not view as part of the problem (or as part of the problem's context), then the additional content is content about how. A technology domain such as communications may be part of the customer's perceived context of the problem. In such cases there may be requirements that dictate the use of a particular communications network. The customer's reasons for this could be organizational and completely independent of technical considerations. But if the customer is oblivious to how information travels from one host to another (or even that there are multiple hosts), then the communications network is part of the solution, not of the problem.

The builder may convince the customer that processing loads necessitate distributing the work over multiple hosts, and that a specific type of communications network is the only viable technology for transferring information within the stated response-time constraints. The customer may even be convinced to weaken the response-time requirements. In such a case the revised performance requirements are still what (this is how design can impact requirements in an iterative development lifecycle), but the communications network is still how because the customer does not especially want the network, he is only willing to accept the network as a solution.

We must now ask into which category the product of our OOS method falls: the problem (what) or the solution (how). Looking back over the seven steps we can see that nowhere is a solution introduced. When we decompose an entity, we ask what the entity does and what it consists of. Both of these questions help define the entity more precisely. This can, potentially, turn into a question of how, if, for example, we decompose the entity into low-level building blocks that belong to some solution domain. This is the point at which analysis should end and design should begin; but not all subentities will be of this sort.

Deciding which functions to group under which entities may seem like a design activity, but it is based on our understanding of what the entity is (and hence what it does). We recast entity data flow diagrams only to achieve a more accurate representation of the entities. If, therefore, we limit ourselves to entities that are part of the problem (not the solution), the recasting of diagrams serves only to better articulate the requirements.

Transition from Analysis to Design

We have developed a design generator tool which reads a hierarchy of EDFDs and outputs a hierarchy of object diagrams. Prior to invoking the generator, the developer is supposed to annotate the EDFDs with some additional information. Default assumptions allow the generator to work even if the annotations are not present. To achieve a reasonable first cut at a design, however, the developer ought to consider the following issues: * Direction of control flow * Concurrent processing threads * Abstract Data Types * Reusable functions

The default assumption for the direction of control flow is that control flow and data flow are in the same direction. If there are data flows in both directions between two entities or functions, this assumption will cause the resulting objects in the object diagrams to call (or use) each other. In most cases this is not the intended result. To indicate that control flow is in the opposite direction of a data flow, the developer annotates the data flow by placing a question mark (?) in front of the data flow label as shown with "Debits" in Figure 15.

The design generator maps entities in an EDFD to ordinary objects in an object diagram. Ordinary objects correspond to Ada packages. The opposite of an ordinary object is a degenerate object. Degenerate objects correspond to Ada tasks and procedures. The highest layer of functions in the data flow model (those lying just below the wavy line in Figure 2) are mapped by the design generator to degenerate objects. Lower-level functions are not mapped into the object diagrams at all. These lower-level functions correspond to subprograms of Ada procedures and tasks. They are not part of the object-oriented view of the design as represented in a set of object diagrams. By default, the highest-level functions are converted into procedure objects. If the function represents an independent thread of control then it should be mapped to a task rather than a procedure. The developer indicates this by placing a pound sign (#) in front of the function name (in the EDFD).

Passive entities in the object-oriented specification are represented as data flows (or stores) with square brackets ([ ]) around the data flow (or store) label. Before executing the design generator, a final check should be made for instances of abstract data types. An ADT that belongs to a Computer Science domain, for example, would not be identified in the OOS. Data flows that are instances of such an ADT would not be marked as entities. To ensure that the ADT does appear in the design, the developer should now annotate the data flow by placing a left-hand square bracket in front of the label. The design generator will create and place the ADT in the top-level object diagram, and will adjust all flows accordingly.

Functions that are potentially reusable (within the present or future systems) are identified by placing an ampersand (&) in front of the function label. Such functions may lie more than one level below the wavy line of Figure 2. The design generator collects all such functions and places them in an object created for this purpose in the top-level object diagram. Flows are adjusted accordingly. This placement of the reusable functions ensures that they are potentially visible to other objects in the system, as well as potentially importable to other systems. We assume that in the final design the single object containing these functions will be split into multiple objects, depending on the nature of the reusable functions. Collecting them into a single top-level object is just an intermediate step, which is taken to ensure their visibility. The object diagrams created by the design generator are intended to serve only as a first cut.

Data Flow vs. Control Flow

Should we have incorporated control flow more fully into the OOS? The question mark notation described in the previous section clearly subordinates control flow to data flow. Control flow is in fact a hybrid concept, encompassing two distinct aspects of a system: * Sequencing of events (flow) * Control hierarchy (who calls whom)

Sequencing is relevant to any method in which functional requirements are analyzed. Sequencing is implicit in a functional decomposition (even in Data Flow Diagrams) because inputs must be available before outputs can be produced. Sequencing is a necessary concept in specifying operational scenarios for a system. Our comments on end-to-end service definition suggest that such scenarios are a necessary complement to OOS (as seen earlier in the "End-to-End Service Definition" section).

The question mark notation does not tell us anything about sequencing. It tells us about a control hierarchy, which may need to be specified in articulating operational requirements. Many Operational Concepts define layers of services by means of a hierarchy of control. The very notion of an object as a provider of services suggests that OOS should enable us to represent such information. Real-time system specification requires a clear definition of stimuli and events, receptors and actors. Including such information in the data flow model would increase the overlap between that model and the state-transition model. This greater overlap between views might promote consistency and soundness of the specification.

We may ask, therefore, why data flow is given predominance in our approach. There are those who argue that while control flow is sometimes important, data flow is always important because functional requirements--that most basic starting point for a requirements analysis--refer to inputs and outputs, which are nothing but data flows. The requirements in their initial (textual) form refer to system inputs and outputs. Internal inputs and outputs arise during functional decomposition, which we have described as a way of clarifying the content of the original functional requirements. Functional decomposition also carries with it the concept of sequence, however, and we have already noted that some amount of sequencing is implicit even in the data flow model.

Our position on this issue is inconclusive. Our own motivation for concentrating on data flow had more to do with practical considerations than methodology. We wanted to use a representation method with which most analysts were already familiar, and DFDs fit that bill. We must admit that focusing on data flow between entities is uncomfortable for us. We do see entities as objects with well-defined interfaces and therefore as service access points. To ignore this during requirements specification feels unnatural to us. The question mark notation does, however, give us a means of specifying hierarchies of control, and for us this is an acceptable compromise. Further use of the method will perhaps confirm our decisions or, if not, then suggest alternatives.

CAPTIONS: Functions are performed by or act on an entity. (chart); An entity-relationship diagram defines the problem domain. (chart); Corrections are suggested by the OOS rule checking program. (chart)

SPECIAL FEATURES: illustration; chart
DESCRIPTORS: Object-Oriented Languages; Requirements Analysis; Analysis; Specifications
FILE SEGMENT: CD File 275

WEST

Generate Collection Print

L3: Entry 15 of 27 File: USPT

DOCUMENT-IDENTIFIER: US 5881238 A
TITLE: System for assignment of work requests by identifying servers in a multisystem complex having a minimum predefined capacity utilization at lowest importance level

Detailed Description Text (7):
Each incoming work request, as well as session request, from any of client computers 10, specifically from client applications 15 running thereon, to the sysplex is accompanied by one or more associated attributes. The request attributes can span a wide variety and are typified by, e.g., user identification (USERID), accounting information, job name or transaction name. For any given request, policy information in dataset 135, using embedded pre-defined rules, is used to map the attributes for that request into a service class. Each different service class has a set of business goals associated therewith. These goals are varied and can constitute, e.g., required response time and required execution velocity--the latter signifying how fast a given piece of work is actually running vis-a-vis how fast that piece of work could run if nothing else was running at the time. Each of these goals has a numeric importance level associated with it which signifies the business importance attached to achieving that particular goal and hence the service class. These levels range from one to five, with one representing the highest importance level, five representing the lowest. Where insufficient sysplex capacity exists to immediately dispatch all work requests then assigned to any given system in the sysplex, those pending requests having higher importance levels prevail and are accorded higher dispatch priority for execution over those requests having lower importance levels. The present invention does not address either the actual prioritizing and dispatch of requests that have already been assigned to a given server or the policy based mapping of the request attributes into proper service classes and importance levels. Rather, as will be discussed in quite some detail below, the present invention is directed to a technique that, based upon business importance of existing requests (active work) and available sysplex resources, actually assigns these requests among the available systems and servers therein for subsequent execution in a manner that satisfies overall business goals of the sysplex, as embodied in the policy. Generally, some of the systems will be running under the goal based policy, while others may not. As will be seen below, our inventive process accommodates both types of systems.

00301123 (THIS IS THE FULLTEXT)

DTC's changing the way it does business: Innovations speed settlement, aid bankers

Anonymous

ABA's Financial Services Industry Trends, v55, p2-3, Mar-Apr 1996
DOCUMENT TYPE: Newsletter Article ARTICLE TYPE: News JOURNAL CODE: BFST LANGUAGE: English RECORD TYPE: Abstract Fulltext
WORD COUNT: 00998

ABSTRACT: Custodians, or agent banks, have a vested interest in being lean, mean, fast, and efficient. The Depository Trust Co. (DTC), principal depository for the US securities industry, handling multi-trillions dollars worth of securities a year, has come up with some new ways to do business that demand attention. They are built upon the interactive Institutional Delivery (ID) system that DTC developed to accommodate T+3. ID is a computer-to-computer message system that moved from an end-of-day, batch processing system that functioned satisfactorily in a 5-day settlement cycle to a continual, intraday, interactive process that already can move faster than T+3. Four new tools are available, including: 1. SID, DTC's Standing Instructions Data Base, 2. Notice of Order Execution (NOE), 3. Institution Instructions (II), and 4. automatic trade matching.

TEXT:

The securities industry adapted to a three-day trade settlement period last year without skipping a heart beat. Now the technological innovation spurred by that mandate has introduced new capabilities that bankers need to know about as the industry moves toward even faster trade settlement. But with all the change, one thing remains constant -- banks are still a critical component of the settlement process.

Custodians, or agent banks, have a vested interest in being lean, mean, fast, and efficient. The Depository Trust Company (DTC), principal depository for the U.S. securities industry, handling multitrillions dollars worth of securities a year, has come up with some new ways to do business that demand attention. They are built upon the interactive Institutional Delivery (ID) system that DTC developed to accommodate T+3.

ID is a computer-to-computer message system that moved from an end-of-day, batch processing system that functioned satisfactorily in a five-day settlement cycle to a continual, intraday, interactive process that already can move faster than T+3. Related developments have the potential to handle T+1- or even T+O. Four new tools are available now, and more are on the way.

Say Hello to SID

SID, DTC's Standing Instructions Database, connects investment managers, brokers, and custodians to a central body of customer relationship information and settlement data that is easily and quickly recorded, maintained, and shared by all parties. Each controls its own information. A correction or notification entered by anyone is available to all automatically. For instance, an investment manager need fill out a new account form only once, and then, with just one key stroke, transmit it to SID for use by all the brokers with which it usually trades. No need to phone or fax the same information to multiple brokers. The data are available to all of them as soon as they are entered. And SID is universally useful. It works equally well for all types of securities and settlement locations, both domestic and international, whether or not they are DTC-eligible.

Key to SID's success is that each party controls its own data. Custodians determine what customer and settlement information they require; investment managers are responsible for all customer account and interested party information and can update it at will; brokers have at their finger tips all the most current settlement data without the responsibility of maintaining it and without having to process last-minute changes. SID creates a win-win-win situation.

Next, NOE

Notice of Order Execution, or NOE, is a simple message designed to give investment managers and broker-dealers the greatest possible speed and accuracy in processing trades. Once a broker-dealer executes a trade, it can use NOE to communicate trade details. Some brokers use NOE for each portion of an order to notify the investment manager of the partial execution price as well as a running average price; others choose to wait until the order has been filled to furnish the average price. The message format accommodates the needs of both trading parties.

NOE may be used with international securities as easily as with U.S. executions. The message format includes fields for securities that settle around the world, both equities and fixed income. Its benefit to bankers is simply that it hastens settlement and makes the process more accurate and efficient.

The Ayes Have It

Institution Instructions, or IIs, are block trade allocation messages -- quick and easy tools investment managers use to respond to NOEs. An II message instructs the broker how to allocate a block trade among accounts or to send the trade data to DTC's new trade confirmation matching facility, or both. It creates a detailed record specific to each allocation that includes all pertinent data, and the format also allows the investment manager to cancel a single allocation within a set. It is remarkably flexible.

Like ID and SID, II works equally well for international and U.S. transactions. There is no restriction as to security type or settlement location.

Although NOE and II were designed to be used together and to enhance SID, they also may be used independently. Universality, efficiency, and flexibility were controlling concepts in their design.

Match Matic

For broker-dealers and institutions opting for automatic trade matching, DTC will compare the broker's trade data to the II allocation. If a match is found, and the institution has the authority to affirm trades, DTC will generate a matched affirm confirmation, thereby shortening the settlement process one step by eliminating the affirmation generally required in response to a confirmation of trade data. If the institution is not authorized to affirm, DTC can still issue a matched confirmation. Then, trade data with settlement instructions from SID can be received, affirmed, and set up for final settlement, all in the same day.

Although SID offers a currency tolerance parameter that allows trades to match even if the amounts are not exact, a match still may fail due solely to settlement amount discrepancies. In that event, DTC sends a Potential Match Report immediately to the broker and to the institution. Failing a potential match, an Unmatched Report goes to institutions and brokerdealers at the end of the day listing all unmatched trades, including items reported on the Potential Match Reports. Unmatched confirmations and IIs are made available to all named parties.

Whether matching or confirmation/affirmation is used, a Cumulative Eligible Trade Report, listing all previously affirmed confirmations, is made available to deliverer and receiver and other parties named on the confirmation on the afternoon of T+2 for settlement the next day.

For agent banks and custodians, these developments at DTC are harbingers of the future as well as most welcome developments in their own right. These enhancements increase speed, control, and accuracy in the trade settlement process -- which is to say they increase efficiency. And even better things are ahead. DTC is moving to a Windows-based workstation, which will greatly enhance the appeal of all these valuable tolls. Additional tools are in the works. To find out more, please call Kathy Kryger at (212) 709-2037.

COMPANY NAMES:

Depository Trust Co DUNS:07-324-9799

CLASSIFICATION: 8130 (CN=Investment services); 5240 (CN=Software & systems); 9190 (CN=United States)

DESCRIPTORS: Custodians; Securities; Clearance & settlement; Technological change; Bank automation

GEOGRAPHIC NAMES: US

?